WalnutDoc.tioga
Willie-Sue on May 6, 1987 3:49:58 pm PDT
Subhana Menis, April 28, 1987 11:23:18 am PDT

HOW TO USE WALNUT

How To Use Walnut

Willie-Sue Orr

Release as [Cedar]<CedarChest7.0>Documentation>WalnutDoc.tioga

© Copyright 1983, 1984, 1985, 1986, 1987 Xerox Corporation. All rights reserved.

Abstract: Walnut is a computer mail system interface that runs in Cedar. It provides facilities to send and retrieve mail (using the Grapevine mail transport system), and to display and classify previously retrieved messages. This document describes how to use Walnut; the document WalnutInterfacesDoc.tioga gives information about programmer access to a Walnut database.

How to Use Walnut: Contents

[If you are reading this document on-line, try using the Tioga Levels menus (if you can) to initially browse the top few levels of its structure before reading it straight through.]

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

For Internal Xerox Use Only

0. Introduction

Walnut is a mail system that runs in Cedar. Walnut provides facilities to send and retrieve mail (using the Grapevine mail transport system), and to display and classify stored (i.e. previously retrieved) messages. Walnut uses the Cypress database system to maintain information about stored messages.

1. Database Structure

We begin with a user's model of Walnut's database. Walnut's database contains two entity types: *message* and *message set*.

A message entity corresponds to a message retrieved from the Grapevine mail transport system. Like all database entities it has a *name*, consisting of the unique message ID provided by Grapevine, expressed as a ROPE of hex characters. A message also has several immutable properties: its *sender*, its *subject*, and so on. Its *unread* property is a BOOL whose value is TRUE when the message is first stored in the database, and is set to FALSE when the message is first displayed.

A message entity is also a member of one or more message sets. A message set entity is named by a text string containing no embedded blanks. There are two distinguished message sets: *Active* and *Deleted*. A newly-retrieved message is made a member of Active. A message that is removed from all other message sets is added to Deleted. Using the Active and Deleted message sets in this way ensures that each message belongs to at least one message set.

Each user will generally have one private Walnut database for reading mail. Additionally, there are public Walnut databases (residing on Alpine servers) that any Walnut user can browse. (Switching from private to public databases is described in Section 4.) You may have either readonly or write access to a public database; if you have write access, then it is possible to move messages among message sets and to delete unwanted messages from the database (by doing an Expunge, as described below).

2. User Interface

Walnut implements four viewer types: the Walnut control viewer, the message set display viewer, the message display viewer, and the message composition viewer. When Walnut is running there is one Walnut control viewer, and any number of instances of the other viewer types. The iconic form of the Walnut control viewer is a mailbox. Anything that can be done with Walnut can be done by starting from the control viewer (sometimes by creating other viewers).

Walnut's user interface attempts to be consistent with conventions used elsewhere in Cedar. Clicking LEFT on a button representing a Walnut entity (a message or a message set) "selects" the entity (makes it an implied parameter to other operations); clicking MIDDLE on such a button opens the entity (displays more information somehow). This is analogous to the behavior of icons in Cedar.

Unless otherwise specified, hereafter "click" means "click with the LEFT mouse button."

2.1 Message sets

The message sets in a Walnut database are represented by buttons in the Walnut control viewer. The Active and Deleted message set buttons always appear first; other message set buttons appear in alphabetical order. There may be several rows of these buttons.

Operations on message sets are performed by using menu items on the second line of the Walnut control window; the MsgSetOps menu item in the first menu line will open a popup menu with several selections. To create a new message set (containing no messages) select its name in some viewer, then click Create (in the popup menu). A message set button with this name will appear and be selected. Most of the other operations in the popup menu use the selected messages sets as implied arguments. To delete one or more existing message sets, select the message sets and click Delete; Walnut requests confirmation if message set to be deleted contains any messages. Make MsgSet(s) Empty will delete all the messages in the selected sets without destroying the sets themselves. To find out how many messages are currently in the selected message sets, click SizeOf; the sizes are printed in the typescript at the bottom of the control window. Clicking PressPrint will print the messages in the selected message sets. Clicking Interpress2.0Print or Interpress3.0Printwill print the messages in the selected message sets, using the relevant version of Interpress. Walnut will load any files necessary for printing.

Finally, the ~~Expunge~~ button removes all of the messages in the Deleted message set and reclaims the storage used for these messages (it "expunges" the Deleted messages). You should expunge when the Deleted message set contains a few hundred messages; when Deleted begins to fill up to the point that it takes several seconds to display, it's time to expunge. Expunging is reasonably fast (a good rule of thumb is that it takes around 30 sec. / 100 messages in the database).

To create a message set display viewer, click MIDDLE on the corresponding message set button of the control viewer. The iconic form of a message set displayer is a stack of envelopes.

The message set displayer contains a one-line button for each message in the set. Each message button looks much like a line in the top window of Laurel: it shows the date of the message, the name of the sender (or To: field if the sender is the current user), and the message subject. At most one of the message buttons in a set is selected (the default is to display it in a small, bold font). To select a message, click LEFT on it; to select and display a message, click MIDDLE on it. To delete a message from the message set, click CTRL-LEFT on it; its button will disappear. (Note that if this message belonged to no other message set, it is added to the Deleted set, and hence is still accessible.)

A message set displayer also contains several command buttons that operate on the selected message:

MoveTo adds the message from the message set to all of the message sets selected in the control window and deletes it from the message set.

Display displays the selected message.

Delete deletes the message from the message set; if it thus becomes a member of no other message sets, then it is added to the Deleted message set.

AddTo adds the messages to the selected sets.

Places is a subset of the Tioga places menu.

Levels is the Tioga levels menu.

MsgOps provides some less frequently used operations (see below).

The Active message set includes a NewMail button that reads new mail and adds the new messages to the Active message set.

The MsgOps (popup) menu provides the following choices:

Sender builds a Sender window

Size of MsgSet reports the number of messages in this message set

Categories lists (in the Walnut control window) the message sets in which the selected message appears (a message can simultaneously be in several message sets).

size Of Msg reports the number of bytes in the selected msg and indicates if it has Tioga formating or not

gvID of Msg prints the database name for the message in the control window.

Append Msg takes the selection contents (or the selected viewer if the length of the selection is less than 2), makes a unique gvID for it and adds it to the message set as a message.

PressPrintMsgSet prints all the messages in the message set (see section 6 for options).

Interpress2.0PrintMsgSet prints all the messages in the message set, using Interpress2.0 (see section 6 for options).

Interpress3.0PrintMsgSet prints all the messages in the message set, using Interpress3.0 (see section 6 for options).

PressPrintSelectedMsg prints the one selected message (see section 6 for options).

Interpress2.0PrintSelectedMsg prints the one selected message, using Interpress2.0 (see section 6 for options).

Interpress3.0PrintSelectedMsg prints the one selected message, using Interpress3.0 (see section 6 for options).

PressPrintTOC prints the TableOfContents of the message set (see section 6 for options).

Interpress2.0PrintSelectedMsg prints the TableOfContents of the message set using Interpress2.0 (see section 6 for options).

Interpress3.0PrintSelectedMsg prints the TableOfContents of the message set using Interpress3.0 (see section 6 for options).

For the MoveTo, Delete, and AddTo buttons, LEFT-clicking simply performs the operation described above, while RIGHT- or MIDDLE- clicking performs the operation and displays the next message in the set. (Note that if you have readonly access to a public database, then the MoveTo, Delete and AddTo buttons will not appear in message set displayers.)

2.2 Messages

As described above, a message can be displayed by clicking MIDDLE on a message button of a message set displayer. This creates a message display viewer, whose iconic form is an envelope (clicking SHIFT-MIDDLE on a message button causes the created viewer to fill the entire column). The message within such a viewer is not editable. This viewer is associated with the message set that created it, so clicking MIDDLE on another message of the same message set shows this new message in the same message displayer. This is designed to avoid a proliferation of message displayers. Of course, there are times when you really want to create viewers on several different

messages in one set. Clicking a message displayer's Freeze button (which then disappears) permanently binds the message to the message displayer. If all message displayers for a given message set are frozen, then MIDDLE clicking in the message set creates a new displayer. A frozen message displayer cannot be unfrozen, but can be destroyed (it will be destroyed when the control window is destroyed).

Answer creates a Walnut Send viewer initialized with a proper heading for an answer.

Forward creates a Walnut Send viewer initialized either with a copy of the message for forwarding.

ReSend creates a Walnut Send viewer initialized with a copy of the message, with the **Date:** field changed to **Original-Date:**, any **Sender:** field changed to **Original-Sender:** and any **From:** field changed to **Originally-From:**.

MsgOps gives a popup menu, with the following selections:

Categories the same as that on message set displayers.

PressPrint print the message using Press.

Interpress2.0Print print the message using Interpress2.0

Interpress3.0Print print the message using Interpress3.0

gvID the same as that on message set displayers.

Split, Places and Levels buttons are from Tioga.

2.3 Sending mail

To create a Walnut Sender, left-click Sender in the Walnut control viewer, or use the Answer or the Forward button on a message viewer, or type WalnutSend to a CommandTool. A Walnut Sender is a Tioga viewer for typing in the header fields and body of a message you want to send.

The Sender menu:

Send is well named. StopSending! pops up during part of the sending process; clicking it gives you a last-minute chance to change your mind about sending a message. Right-clicking Send makes the Walnut Sender become iconic after syntax checking. (In iconic form a Walnut Sender is the back of an envelope.) A Walnut Sender will not allow itself to be destroyed while sending is in progress.

SendChecked will warn you if the message to be sent contains comment nodes; if it does, you have the option of sending anyway or quitting.

Get is similar Tioga Get -- Will load a local or remote file and allow editing

Store is like Tioga Store. It will leave the sender in edited mode, unless you used right-click.

Save is like Tioga Save. It will leave the sender in edited mode, unless you used right-click.

Forms shows the Forms popup menu, which provides three options plus forms supplied by the user..

NewSender creates another Sender.

DefaultForm replaces the current Sender with the default. (You can specify a non-

standard default in your User.profile.)

Previous Msg restores the last message successfully sent from the current Sender.

You specify your own forms (see below).

Get and Store change the name of the Sender to be the file name. Clicking the Forms items will cause any backing file association for the Sender to be cleared and the caption will revert to Walnut Sender.

Split is the Tioga Split.

Places is the Tioga Places.

Levels is the Tioga Levels.

The default Sender form has node structure and uses a small bold font (looks) for the header fields. Added header fields use the same looks; to get added fields to have some other looks, include in your User.profile the key WalnutSend.MsgHeadersLooks, set to whatever you like.

If you want to change the default forms, you can create your own and specify them in your User.profile with the entries:

WalnutSend.DefaultForm: filename

WalnutSend.AnswerForm: filename

WalnutSend.ForwardForm: filename

The answer form must have header lines in the same order as the standard form and provide placeholders in the appropriate header positions. The forward form must have a node with the contents "ForwardedMessage"; Forward replaces this node with the message being forwarded. These private defaults can be local or remote.

If you need additional sending forms, you can specify them in your User.profile with the entry:

WalnutSend.MsgForms: {list of file names}

These additional forms will then appear in the pop-up Forms menu. Like private defaults, these additional forms can be local or remote. (You might, for instance, want to specify remote forms from /Cedar/CedarChest®/Top/Forms.df.) Only the short name pops up: the default extension is .form, and only non-default extensions pop up. At the moment, there is no way to create additional answer or forward forms.

A message to a public distribution list or to more than twenty individuals probably needs a Reply-to: field. If your message doesn't contain one, Send will pop up a menu with the items Self, All, and Cancel. Self means insert the Reply-to: field and fill in my name; All means do not insert a Reply-to: field (replies then go to the full list); Cancel means whoops, don't send the message yet. To get Reply-to: sender when appropriate, without being asked, insert the entry

WalnutSend.ReplyToSelf: TRUE

in your User.profile. (With this entry, if at some point you want Reply-to: all you have to insert it by hand.)

Watch the Walnut control window for errors in transmission. If the transmission is successful, the message is saved and a new default form appears.

WalnutSend will also run standalone, in which case it creates a small typescript viewer for reporting progress, errors, etc.

2.4 Retrieving mail

Walnut polls the mail servers at regular intervals. If there is new mail for the logged-in user and the database being read is the user's private database, the mail is retrieved from grapevine and stored on the user's newMailLog on Alpine. The Walnut control viewer will then display a message like

> You have new mail at May 28, 1985 6:13:43 pm PDT

(if you are reading a public database, the message will simply be

> Cannot retrieve mail using this database (if you have write access) or
> You only have Read access to this database).

If the control window is iconic and there is new mail, the flag on the icon is raised.

Clicking the NewMail button (which will not be present if you're browsing a public database) in either the control viewer or the Active message set viewer retrieves all the new mail into the database. The new messages appear as buttons at the bottom of the Active message set viewer, displayed with the looks to be used for unread messages.

Walnut will automatically copy the new mail from the newMailLog and add it to the database for you. To enable this feature, include:

```
Walnut.AutoNewMail: TRUE
```

in your profile. During startup or restart, Walnut may not immediately notice that there is new mail on grapevine, even though the status line says there is; it will take Walnut at most 1 minute to do the automatic retrieve, unless some long running operation is in progress.

2.5 Queries

See WallabyDoc for details.

2.6 Global operations

The main menu of the Walnut control window provides the following buttons:

Sender creates a WalnutSend viewer

NewMail retrieves mail from the newMailLog

CloseAll button furnishes a quick way of ending a session with Walnut: it destroys all message displayers, and closes all message set displayers and the Walnut control viewer.

MsgSetOps opens a popup menu (see below).

Find does a Tioga-style find operation on the MsgSetButtons subviewer.

~~Expunge~~ (guarded) expunges the Walnut database: all messages in the Deleted set disappear without a trace. Without the Expunge operation, the database and log would grow without bound. Section 4 contains more information on this topic.

The MsgSetOps popup menu provides the following operations:

DB-Info prints the length of the log, the number of deleted messages and the number of messages and message sets in the typescript of the control window.

SizeOf, PressPrint, Interpress2.0Print, Interpress3.0Print, Create, Delete, Empty as explained

R.O.T. (short for Remove Over There) removes all messages in selected message set from other message sets they participate in.

Archive provides a way to copy a set of messages into a file that can later be read by another mail system that uses the "PARC standard mail format" (Hardy and Laurel do). Select a filename in some window and click Archive. All messages in the currently selected message sets are copied to the named file (if the file name has no extension, ".ArchiveLog" is appended to the name).

Append same as Archive but appends the messages to the given file.

ArchiveAndDelete, AppendAndDelete do the archive or append and then delete the messages in the messages sets (asking for confirmation).

Walnut registers several infrequently-used commands with the Cedar Executive. All of the command names contain the prefix "Walnut", so typing Walnut*? to the Executive enumerates them. The Walnut command creates a Walnut control viewer if you should happen to Destroy yours; to open a public database provide the Walnut root file name as an argument (e.g., Walnut [Luther.Alpine]<CSL-Notebook>Walnut.root opens up the database defined by [Luther.Alpine]<CSL-Notebook>Walnut.root). The WalnutExpunge command has the same effect as ~~Expunge~~. The WalnutOldMailReader command is described in Section 3, and the WalnutScavenge command is described in Section 4. WalnutNewMail simulates clicking the NewMail button. The WalnutSend command is like clicking Sender. WalnutScavenge is discussed in Section 5.

In addition, there are several CommandTool commands to control printing, allowing one to specify some server other than the default, or to make more than one copy (currently only for printing Press files). In the following, <server> and <copies> may be elided, but you must specify a server if you wish to make more than one copy. In addition, the server named "*" causes the file to be written but not printed. The commands are (the IPx versions create Interpressx masters):

WalnutInterpress2.0PrintMsg, WalnutPressPrintMsg msgID <server> <copies>

WalnutInterpress3.0PrintMsg, WalnutPressPrintMsg msgID <server> <copies>

prints the named message (get its msgID by clicking gvID and copying the name from the control window)

WalnutInterpress2.0PrintMsgSet, WalnutPressPrintMsgSet msgSet <server> <copies>

WalnutInterpress3.0PrintMsgSet, WalnutPressPrintMsgSet msgSet <server> <copies>

WalnutInterpress2.0PrintMsgSetTOC, WalnutPressPrintMsgSetTOC msgSet <server> <copies>

WalnutInterpress3.0PrintMsgSetTOC, WalnutPressPrintMsgSetTOC msgSet <server> <copies>

WalnutInterpress2.0PrintSelected, WalnutPressPrintSelected msgSet <server> <copies>

WalnutInterpress3.0PrintSelected, WalnutPressPrintSelected msgSet <server> <copies>

prints the selected message in the named message set

3. The Log

Walnut keeps a record of all retrieved messages and all database updates for a single user in a *Walnut log file.* This is a text file with a very simple format; you can load your Walnut log into a Tioga viewer to see this. (You will have to use Yodel to copy your log from alpine to do this.)

The important point is that the truth about your Walnut mail resides in a Walnut log file; the Cypress database that Walnut uses for query processing can always be reconstructed by replaying a Walnut log file. The Walnut log mechanism is robust, making Walnut's mail storage reliable even if Walnut (or some other part of Cedar) crashes.

Walnut uses several log files. The *current log* is the one on which messages are stored and the database updates recorded. The *expunge log* is the log written when an expunge is performed (this log becomes the current log when the expunge completes). The *new mail log* is used to copy new messages from the Grapevine servers; a separate log is used so that mail retrieval can be performed simultaneously with normal operation. When the user does a NewMail operation, the contents of the new mail log are copied to the current log. Finally, the *read archive log* is used to copy the contents of Walnut archive files (when performing a WalnutOldMailReader operation); a separate log is used here to ensure that the current log is changed only if the archive file can be successfully read and parsed. (**Note**: currently all of these files must reside on the same Alpine server.)

To provide a connection among the various log files (and the Walnut database), a Walnut *root file* is used: the root file contains the names of the various log files, the name of the database file, and additional information (like the Grapevine RName of the user who may retrieve new mail for this database). Walnut locates the root file for a particular user by consulting the Walnut.WalnutRootFile entry of the user profile; if there is no such entry, the name "[luther.alpine]<*UserName.registry*>Walnut.Root" is assumed. An example of a root file is given below:

```
-- A template for a Walnut.Root file - Template-Walnut.root
-- Copyright (C) 1985 by Xerox Corporation.  All rights reserved.
-- Willie-Sue, May 15, 1985 6:53:12 pm PDT

-- NO TIOGA FORMATTING ALLOWED

-- the Entry types: Key, MailFor, Database, NewMailLog, ReadArchiveLog must appear
-- there must be two and only two LogInfo entries, one with a 7-digit sequence number,
-- the other with the 7-character word Expunge.

-- End must be the last entry and must appear.
-- Blank lines and comments may appear, except in the Key entry

-- ALL files MUST be on the same Alpine server

-- The Key is stored in each log file and the database
```

```
-- It is checked when Walnut opens any of the files named in the root

Key
User supplied key

MailFor
UserRName     -- rname, including registry if a real name

-- database name
Database
▶ServerAndDirectory◀Walnut.segment

NewMailLog
▶ServerAndDirectory◀Walnut.NewMailLog

ReadArchiveLog
▶ServerAndDirectory◀Walnut.ReadArchiveLog

-- the first log file
LogInfo
▶ServerAndDirectory◀Walnut.LogX
100001
0000001

-- the first expunge log
LogInfo
▶ServerAndDirectory◀Walnut.LogY
100002
Expunge

End
```

Please note: *Do not delete or otherwise modify the files specified in your root file unless you are sure you know what you're doing.*

If you need to do major surgery on your Walnut files, read the WalnutRescue documentation for a description of a collection of programs to be used to recover from major disasters.

4. Becoming a User

First, bring over the latest Walnut from the current release directory. This will also retrieve WalnutSend, Cypress, AlpineUser, and various programs for printing. These latter files will be also loaded by Walnut. You can type OpenR WalnutConversion.tioga to the commandTool for more information.

Before you can run Walnut, you will also need an account on some Alpine server to store your Walnut files. If you're in Palo Alto, contact Ron Weaver to obtain an Alpine account; tell Ron you are using Walnut and he will arrange to have your log files backed up. Otherwise, contact

your local Alpine Administrator.

Next, run the command CreateWalnutWorld. It will register two commands, CheckAndInit and InitWalnutFiles and will print a long comment about how these commands are to be invoked. You will want to run InitWalnutFiles, unless you were a previous Walnut user and want to copy your old Walnut mail database into the new system. In that case you should do an OpenR WalnutConversion.tioga and read carefully.

Edit your personal profile to contain all of the entries specified in WalnutDefault.profile (public in Walnut.df). The only profile entry that most users will want to experiment with is "InitialActiveRight: TRUE"; making it FALSE causes the Active message set displayer to create itself in the left viewer column, like all other message set displayers.

To start Walnut, type

```
Walnut
```

to a CommandTool. This will spend a long time loading, but finally a Walnut control viewer will appear.

You can include Walnut in a checkpoint. Be sure not to click checkpoint until the message "...Ready" appears in the Walnut control viewer typescript. Message and Message set displayers get updated after each rollback; if a displayed message or message set has since been deleted, the viewer will be destroyed.

To read a Laurel or Hardy mail file, or a file created by Walnut's Archive operation, first run Walnut as just described. Then type

```
WalnutOldMailReader <complete mail file name> {optional message set name}
```

to a CommandTool. If you fail to specify a message set name, the messages will be placed in Active. If the specified message set does not exist, it will be created. Using WalnutOldMailReader to read a Walnut archive log will put the messages back into the message sets they were in when archived.

5. Coping with Releases and Crashes

Walnut sometimes crashes because its database has gotten into a bad state. Also, a new release of Walnut or the Cedar database system will occasionally change the database format that Walnut understands. From Walnut's point of view these circumstances are very similar.

A Walnut database can always be reconstructed by replaying a Walnut log file. If Walnut is not loaded, you can reconstruct the Walnut database by executing the command WalnutScavenge; this will rebuild the database from the log file, and leave Walnut running when done. Even if Walnut is already loaded, you can scavenge by typing WalnutScavenge to the Executive. Walnut will also scavenge automatically if the database cannot be found.

6. User Profile Options

Below is a complete list of all of the current Walnut user profile options:

Walnut.AutoNewMail: BOOL ← FALSE;
if TRUE, then automatically retrieves new mail when the newMailLog is nonempty.

Walnut.AutoScrollMsgSets: BOOL ← TRUE;
if TRUE, automatically scrolls a msgset (when first displayed) to the first unread message (if any) or to the end.

Walnut.DefaultArchiveDir: TOKEN ← "";
value is the default working directory when reading or writing archive files.

Walnut.DisplayMsgSetInIcon: BOOL ← FALSE;
If TRUE, the mailbox icon contains a label showing the name of the currently-selected message set.

Walnut.GuardedDestroy: BOOL ← FALSE;
if TRUE, then the Destroy button in the Walnut Control window will be guarded.

Walnut.InitialActiveIconic: BOOL ← FALSE;
if true and InitialActiveOpen = TRUE, then the Active message set viewer is opened as an icon.

Walnut.InitialActiveOpen: BOOL ← FALSE;
true says open a message set viewer on Active.

Walnut.InitialActiveRight: BOOL ← TRUE;
true says to bring up the active message set on the right column. false on left.

Walnut.InterpressPrinter: TOKEN ← "Quoth"
The printer to use for Interpress files.

Walnut.MsgSetButtonDefaultLooks: TOKEN ← "";
the looks for unselected MsgSetButtons.

Walnut.MsgSetButtonSelectedLooks: TOKEN ← "bi";
the looks for selected MsgSetButtons.

Walnut.NewPageEveryMsg: BOOL ← FALSE;
if TRUE, when printing a msgset, every message will begin on a new page.

Walnut.NumMsgSetButtonsLines: INT ← something reasonable
calculated to be somewhat reasonable, max 10 if not specified

Walnut.PlainTextStyle: TOKEN ← "cedar"
The Tioga style to use for message which don't have any Tioga formatting in them.

Walnut.PrintSmallHeaders: BOOL ← TRUE;
if TRUE, the headers sections of msgs get printed in a smaller font. This is NOT used for the print button on a message displayer.

Walnut.ReportNewMailProgress: BOOL ← FALSE;
if TRUE, opens a small typescript and reports as mail is retrieved from grapevine.

Walnut.ShowUnreadWithQMs: BOOL ← TRUE;
if TRUE, shows unread messages with a ? in front: setting this FALSE moves the TOC entry over a couple spaces.

Walnut.TOCDefaultLooks: TOKEN ← "";
the looks for an unselected table-of-contents entry in a msgSet displayer.

Walnut.TOCSelectedLooks: TOKEN ← "sb";
the looks for the selected table-of-contents entry in a msgSet displayer.

Walnut.TOCUnreadLooks: TOKEN ← "i";
the looks for an unread table-of-contents entry in a msgSet displayer.

Walnut.UseFromFieldInTOC: BOOL ← FALSE;
use the From field in the TOC entry, even if there is a Sender field

Walnut.WalnutRootFile: TOKEN ← "[Luther.Alpine]<UserName.registry>Walnut.Root";
value is the name of the file to be used for the Walnut root file.

WalnutSend.AlwaysOpenSender: BOOL ← FALSE;
if TRUE, *the first time WalnutSend is typed to the exec, the sender will be opened and grab the input focus.*

WalnutSend.AnswerForm: TOKEN ← "";
The file name of the answering WalnutSend form (used when Answer is clicked in a message viewer).

WalnutSend.DefaultDLDir: TOKEN ← "";
value is the default working directory for private DL files.

WalnutSend.DefaultForm: TOKEN ← "";
The file name of the default WalnutSend form (used when NewForm is clicked in a Sender).

WalnutSend.DestroyAfterSend: BOOL ← FALSE;
if TRUE, *causes sender to be destroyed after a successful delivery, if Send was clicked with RIGHT*

WalnutSend.ForwardForm: TOKEN ← "";
The file name of the forwarding WalnutSend form (used when Forward is clicked in a message viewer).

WalnutSend.MsgForms: LIST OF TOKEN ← "";
A list of file names to be used as Walnut message forms (buttons for each file listed are added to WalnutSend viewers).

WalnutSend.MsgHeadersLooks: TOKEN ← "sb";
The looks to be used for the labels for the added headers of sent messages

WalnutSend.ReplyToSelf: BOOL ← FALSE;

if TRUE, *causes Walnut to automatically supply a Reply-To: field, if appropriate.*

7. Shortfalls and Wishes

What follows is a listing of known deficiencies and contemplated extensions to Walnut. Nobody guarantees that everything listed below will be implemented. But the list does indicate some directions for future work, and may provide context for your own Walnut wishes. Send both bug reports and wishes to WalnutSupport↑.

7.1 Message sets

It would be nice to allow selection of more than one message in a message set (perhaps even spanning message sets).

7.2 Retrieving mail

Now that mail retrieval is implemented in the background, a natural next step is to provide some means for a user procedure to classify incoming mail according to its significance, file it in sets other than Active, let the user know the status of his new mail ("You have *important* new mail").

The procedure that stores new mail in the database should understand the In-Reply-To relationship. Eventually, users should be able to write queries or other commands that exploit this relationship.

7.3 Sending mail

It should be possible to forward or answer multiple messages. This requires the ability to select multiple messages.

When sending a sequence of messages with the message composition viewer, you tend to click Send, wait for the feedback "sending ...", then make the viewer iconic (to reclaim the screen area) and finally click Sender to create a new viewer. It would be smoother to reuse the same message composition viewer, but without waiting for the message to be sent (since this can take quite awhile). Since it is quite unusual to have two messages in transit (as contrasted with two messages being composed) at the same time, this can be achieved by passing responsibility for the message from the message composition viewer to the Walnut control viewer when message parsing is complete, and clearing the composition viewer for reuse. Any errors in transmission would be reported in the control viewer rather than the composition viewer.